DIEGO SOUSA SANTOS – 11044616



RELATÓRIO EP-GOSSIP – SISTEMAS DISTRIBUÍDOS

SÃO CAETANO DO SUL – SP

10/2022

LINK DO VÍDEO: <https://youtu.be/hI4q6p4p0vo>

1. FORMATO DA MENSAGEM TRANSFERIDA

As mensagens são transferidas através do objeto Mensagem, descrito no diagrama UML abaixo:

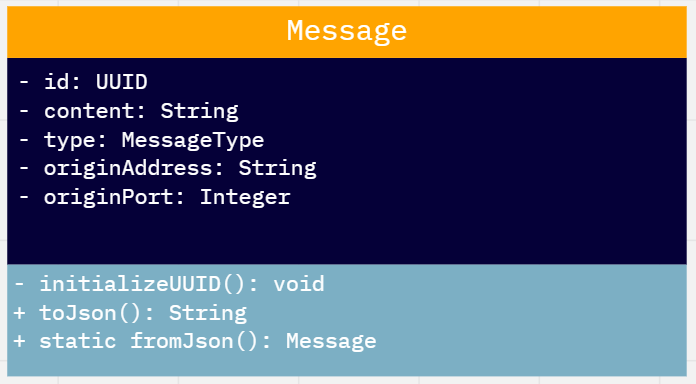


Figura UML da classe Message

O atributo “id” define o identificador da mensagem através de um UUID que permite tornar única cada mensagem.

O atributo “contente” armazena o conteúdo da mensagem. No nosso caso, o arquivo que está sendo procurado.

O atributo “type” é um struct que classifica a mensagem como SEARCH ou RESPONSE, possibilitando diferenciar esses dois tipos de mensagens.

O atributo “originAddress” armazena o endereço IP do Peer de origem da mensagem.

O atributo “originPort” armazena a porta do Peer de origem da mensagem.

O método “initializeUUID” gera o UUID da mensagem.

O método “toJson” utiliza a biblioteca gson para converter o objeto Mensagem em Json.

O método estático “fromJson” retorna uma Mensagem a partir de um json.

1. PEERS E SUAS FUNCIONALIDADES

2.1 MÉTODO MAIN

O método main é definido logo no início da classe Peer. Nele existe um loop para leitura dos dados do usuário. É impresso um menu composto das opções “1. INICIALIZA” e “2. SEARCH”, as quais o usuário pode selecionar informando o número ou o texto da opção.

Caso a opção de inicializar seja selecionada, o usuário deverá informar o endereço IP, a porta, o diretório de arquivos e duas portas de peers vizinhos para o novo peer que será criado. Com isso, uma nova instância de um objeto peer é criada e manipulado nas consultas de search.

Caso a opção de search seja selecionada, o usuário deverá informar o nome do arquivo que será buscado na função de search.



Figura código para inicializar peer

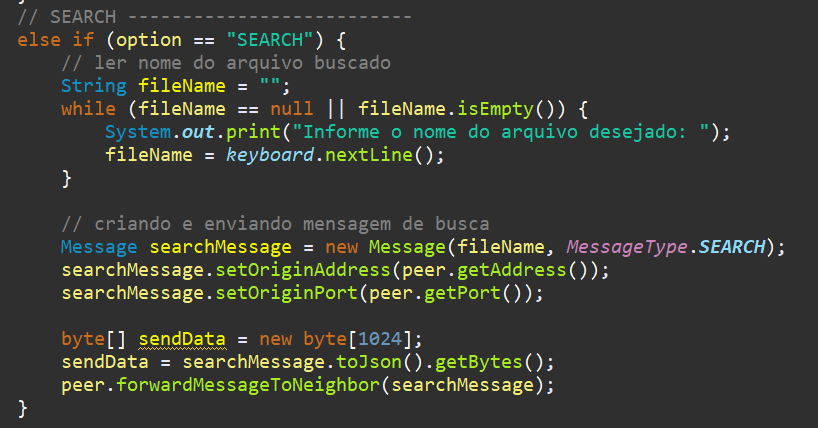


Figura código para buscar arquivos em peers

2.2 DEFINIÇÃO DO PEER

Os peers são definidos através da classe descrita no diagrama UML abaixo:



Figura diagrama UML do Peer

2.2.1 ATRIBUTOS

O atributo “port” define a porta em que o socket do peer é criado, enquanto o atributo “address” define o endereço do socket.

O atributo “filesFolder” armazena o diretório de arquivos do peer, enquanto o atributo lista “filesPaths” armazena os nomes dos arquivos desse diretório.

O atributo “neighbors” registra as portas dos peers vizinhos ao peer em questão.

Os objetos “timerFiles” e “timerTimeOut” são cronômetros assíncronos (via thread” para realizar as rotinas de atualizar a lista de arquivos e calcular timeout, respectivamente.

A lista “receivedMessagesControl” faz a gestão das mensagens recebidas para evitar duplicidade de tratamento. Já o Map (ou dicionário) “sentMessagensControl” faz a gestão das mensagens enviadas, armazenando o horário do envio para posterior tratamento de timeout.

O objeto “serverSocket” é o DatagramSocket utilizado pelos peers para se comunicar, e a constante TIMEOUT\_SECONDS define o limite máximo de tempo em segundos para determinar o timeout.

2.2.2 MÉTODOS

O método closeSocket() apenas realiza o fechamento do socket de um peer.

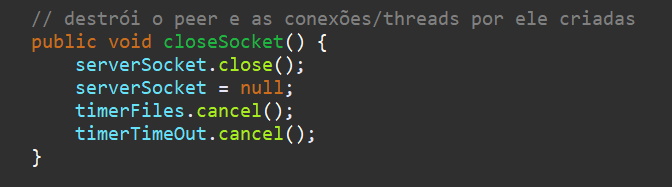


Figura método closeSocket()

O método createSocket(), por sua vez, inicializa o serverSocket e cria um loop assíncrono (linhas 211 à 278) para receber as diversas requisições que podem chegar simultaneamente. Quando uma mensagem é recebida em formato de json, ela é convertida em um objeto mensagem e é realizada uma verificação na lista receivedMessagesControl para garantir que a mensagem já não está sendo tratada pelo peer. Caso contrário ela é descartada.

Depois disso, caso a mensagem seja do tipo SEARCH, é verificado se no diretório do peer existe o arquivo informado no contente da mensagem. Se o arquivo existe, é enviada a resposta ao peer de origem através do método answerToOriginPeer. Caso contrário, a mensagem é redirecionada através do método forwardToNeighbor.

Caso a mensagem seja do tipo RESPONSE, o peer exibe na tela que o arquivo foi encontrado e a mensagem é removida do controle de mensagens enviadas.

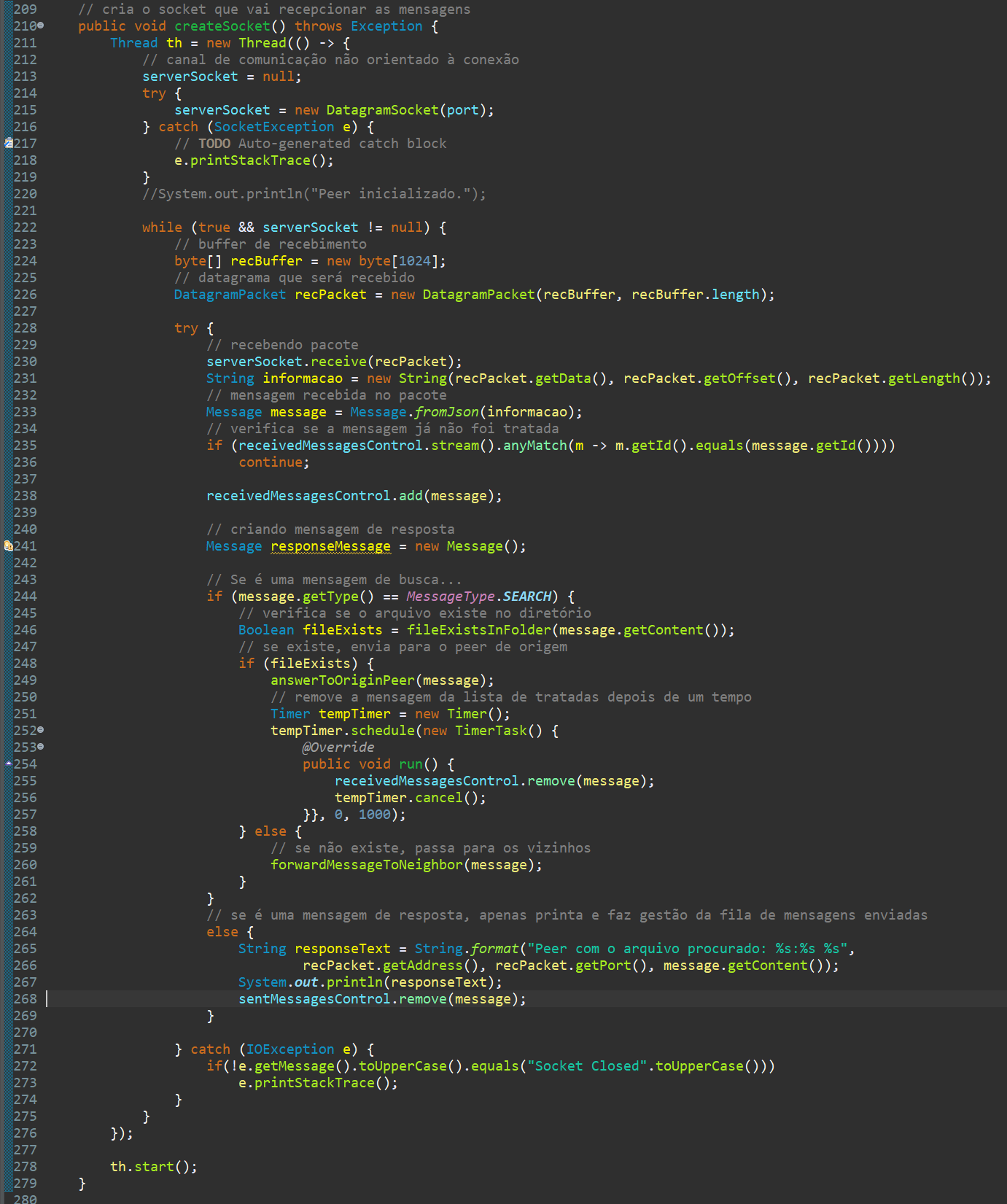


Figura método createSocket()

O método answerToOriginPeer() é responsável por criar uma nova mensagem de RESPONSE utilizando o endereço e a porta de origem da mensagem de SEARCH para sinalizar ao peer de origem que o arquivo foi encontrado.



Figura método answerToOriginPeer

O método forwardMessageToNeighbor() é responsável por criar uma thread (linha 306 ) que selecionará aleatoriamente os peers vizinhos (linhas 314 à 316) para encaminhar a mensagem de SEARCH para eles, desde que não sejam o peer de origem da mensagem. A partir dessa seleção, é criado o datagrampacket para transportar a mensagem e ela é retransmitida ao próximo peer e adicionada ao controle de mensagens enviadas (linhas 334 à 346).



Figura método forwardMessageToNeighbor

O método fileExistsInFolder() verifica se o arquivo solicitado na mensagem existe no diretório do peer. Para isso, esse método concatena os nomes dos arquivos em uma string, separados por espaço, e verifica se o nome do arquivo procurado está nessa string.

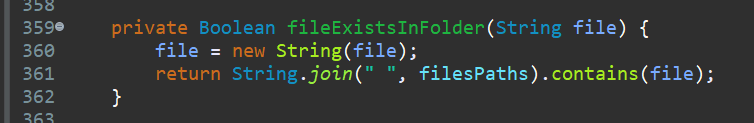


Figura método fileExistsInFolder

O método addNeighbors adiciona os peers vizinhos informados pelo usuário à lista de vizinhos do peer em questão.

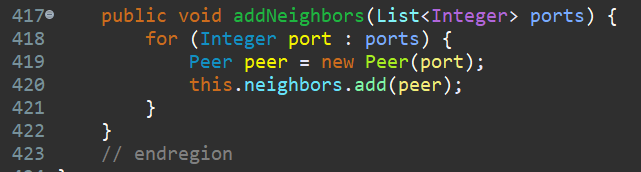


Figura método addNeighbors

Os temporizadores citados anteriormente são inicializados no construtor do Peer. O timerFiles faz o scheduler de uma função que lê os arquivos do diretório do peer, preenche a lista filesPaths e exibe na tela a mensagem “Sou peer x:y com os arquivos z”.

O timerTimeOut faz o scheduler de uma função que a cada segundo visita o map sentMessagesControl para procurar mensagens que foram enviadas a TIMEOUT\_SECONDS segundos ou mais, calculando isso a partir da data de envio da mensagem. Caso encontre, retira a mensagem dos controles e alerta sobre o timeout.



Figura definição dos timers